REMARKS

Request for Withdrawal of Finality of Rejection

In the last Office Action, claims 1 and 28 were rejected on new formal grounds for reasons that relate to language that appeared in the original claims. Therefore, these new grounds of rejection could not have been based on claim amendments and therefore could not have been made final.

Moreover, the claims have been rejected for the first time on new prior art and the Examiner has not even alleged that these new grounds of rejection were necessitated by the claim amendments. Furthermore, it is believed that the previous claim amendment did not, in fact, justify the new grounds of rejection.

Accordingly, it is requested that the finality of the rejections presented in the previous Action be withdrawn and that, therefore, the present Amendment be entered as a matter of right.

Response to Office Action

With regard to item 1 on page 2 of the Action, a replacement Abstract is submitted herewith, even though there has been no objection thereto.

In response to the requirement presented in section 2 of the Action, submitted herewith is a substitute Specification in which double line spacing is employed.

With regard to the objection presented in section 3 of the Action, the language in question has been added in the listing of claims, forming part of this Amendment. It must be noted, however, that although the Examiner has correctly quoted MPEP §608.01(m), the requirement stated in the Manual is intended to indicate the grammatical form of a claim, and does not specify that the exact words "I claim" must be present. Indeed, that section of the Manual specifies the procedure for providing such words if they are not present in the Application and does not indicate that the applicant should be required to add that language.

It is therefore requested that the objections presented in the Action be reconsidered and withdrawn.

In response to the formal rejection presented in section 5 of the Action, the reference to "the host processor" has been removed form lines 20-21. Claim 1 continues to include a positive recitation of the host processor.

The phrase "the user selected parameters" has been deleted from claim 1 and the term "conventional" has been deleted from claim 28.

In addition, the claims have been amended to further define the contribution of the invention over the prior art, and to ensure consistency between the independent claims and the dependent claims.

The rejection presented in section 7 of the Action is respectfully traversed for the reason that the invention, as now defined in the rejected claims, and particularly in independent claim 1, is not disclosed in the applied reference.

The Application claims have, from the outset, been directed to a pre-amplifier apparatus and are now limited to apparatus including "reconfigurable circuit means which are configured in real time under control of configuration data".

The applied reference, O'Brien, discloses power amplifiers, which are different in structure and function from pre-amplifiers.

Power amplifiers and pre-amplifiers are very different from one another and perform very different functions. In fact, in the present application specifically mentions that the pre-amplifier apparatus according to this invention performs signal processing and conditioning before being output to a power amplifier.

A power amplifier receives a signal from a pre-amplifier and amplifies so that it can drive an array of loudspeakers. This is completely different from a pre-amplifier as described in the present application in which stored digital data (of which there are several formats) is decompressed, decrypted, and filtered and then post-processing is performed. The latter operations include surround sound processing and bass management. O'Brien's power amplifier is an extension of the Class D principle i.e. performing as much of the power amplification operation in the digital domain.

The only reconfigurable devices disclosed in the applied reference are known as FPGA's which are incapable of being reconfigured in real time. Although the present Application does mention such devices, among a number of others, the claims of the present Application have been limited to devices that can be configured in real time, and thus to exclude devices of the FPGA type.

FPGA's are by definition field programmable and have to stop processing (be taken off line) and be reprogrammed with the relevant reconfiguration data. This can take many thousands of clock cycles. This is confirmed by the following information provided in a Xilinx datasheet, pages 1-16 of which are attached hereto, which confirms that FPGA's cannot be reconfigured in real time:

1). As described in the flow diagram of Figure 10, on page 12, and point 2 of the section entitled "Start up" on page 13, the Input and Output (IO) pins of the device are tristated (Three state). Therefore, these IO pins are inoperable and so there can be no communication with any other device. The FPGA is therefore not performing any application tasks.

2). As described in the flow diagram of Figure 10 and the section entitled "Clearing Configuration Memory" on page 13, all the configuration memory is cleared at the start of the configuration process. Consequently, an FPGA can not then be performing any logic or arithmetic functions.

3). Table 7 shows the size of the configuration files to configure various types of FPGA's. It should be noted that the larger the FPGA, the longer it takes to configure the device. Therefore, more complex systems will take longer to configure and these larger devices will more likely be used to implement applications as the smaller devices are used for peripheral "glue logic."

4). The configuration operation is described in the datasheet. Data can be loaded serially and byte parallel. However, for the smallest FPGA (XC2S15 device), the configuration file is 197696 bits. The configuration file for the largest FPGA (XC2S200) is 1335840 bits. Data can be loaded with a 60MHz clock, though the first 60 bytes use a 2.5 MHz clock and the default clock is 4MHz (please see page 16).

Assuming the fastest rate clock rate was employed using the smallest FPGA (XC2S15), then it would take at least 3.29 mSec to load the configuration data serially and 0.412 mSec byte-wise.

For the largest FPGA (XC2S200) it would take at least 22.26 mSec to load the configuration data serially and 2.78 mSec in byte-wise mode.

5). After the configuration data has been loaded, there is then a start-up procedure, which also takes a finite amount of time.

6). Note, to process basic speech (3.4KHz) digitally, the sampling rate is 8K samples per second or a sample every 0.125 mSec. If a XC2S15 FPGA was employed to process speech and was required to be reconfigured to implement different digital signal processing algorithms, then in 0.412 mSec (the fastest time for configuration) it would have missed at least three speech samples. It would also be wasteful to use a 60MHz clock in this application just to load new data when the real application would require a clock of a lower frequency. Employing two clock circuits would also be costly and wasteful.

As described in the accompanying FPGA datasheets from Xilinx, an FPGA cannot operate while it is being configured. The I/O is tri-stated during the configuration stage and the device needs to go through an elaborate initialization routine before becoming functional again. Therefore, FPGA devices can't be configured in real time at a rate to implement different high-speed digital signal processing functions.

For many music applications the basic sampling rate is 44.1K samples per second (CD quality) or one sample every 0.0226 mSec. Consequently, it would not be possible to reconfigure an FPGA in real time to implement different digital signal processing algorithms or sub-functions. It takes too long to configure the device and many data samples would be lost.

Thus, FPGA's inherently have serious limitations that the present invention obviates by using reconfigurable circuit means that can be configured in real time, as defined in claim 1 of the present application, to implement different functions. Apparatus according to the invention proves to be particularly advantageous in cases where audio processing algorithms are sequential in nature i.e. one sub-function or operation needs to be performed before the next. In the past, designers have designed an individual circuit for each sub-function or operation and then implemented all of the circuits on an ASIC device. However, by providing hardware circuitry that can be reconfigured in real time to implement the sub-functions as they are needed, the amount of circuitry is reduced. This then reduces the ASIC cost and overall system costs. It also allows the same programmable device to be reconfigured in real time to implement different audio processing standards, digital rights management schemes and post processing algorithms. Hence, this different type of programmable logic device obviates the limitations of FPGAs.

None of the cited references mentions real time reconfiguration or discloses a device capable of being reconfigured in real time.

It is therefore submitted that claim 1, as well as all the claims dependent therefrom, now distinguishes patentably over the applied reference, at least by its recitation of a "reconfigurable circuit means which are configured in real time under control of configuration data".

All of the other prior art rejections are traversed for the reason that the rejected claims depend from claim 1, and should be considered allowable along therewith.

Moreover, a number of these claims define further novel features of the invention that are not disclosed in the applied references.

For example, claim 8 specifies that the apparatus is configured for simultaneous use by more than one user where signal data from one or more signal sources is processed and output to one or more output circuits.

The rejection of claim 8 was based on the view that it would be obvious that one or more users may listen to the output of a speaker. However, the claimed apparatus allows the processing of signals from one or more sources at essentially the same time, allowing more than one user to access the same piece of equipment. This is another advantage of the real time reconfiguration of the logic circuits.

Normally, as cited by O'Brien, two users would require two separate pieces of equipment to process two different signal sources simultaneously. It is therefore submitted that claim 8 is not suggested by, and hence not obvious, over the applied references.

If the above amendment should not now place the application in condition for allowance, the Examiner is invited to call undersigned counsel to resolve any remaining issues.

Respectfully submitted,

BROWDY AND NEIMARK, P.L.L.C.
Attorneys for Applicant

By [illegible]
Jay M. Finkelstein
Registration No. 21,082

JMF:dtb
Telephone No.: (202) 628-5197
Facsimile No.: (202) 737-3528
G:\BN\H\Hasl\Smith11\PTO\AMD 12APR05.doc

Spartan-II 2.5V FPGA Family: Functional Description

DS001-2 (v2.1) March 5, 2001

Preliminary Product Specification

Architectural Description

Spartan-II Array

The Spartan-II user-programmable gate array, shown in Figure 1, is composed of five major configurable elements:

- IOBs provide the interface between the package pins and the internal logic
- CLBs provide the functional elements for constructing most logic
- Dedicated block RAM memories of 4096 bits each
- Clock DLLs for clock-distribution delay compensation and clock domain control
- Versatile multi-level interconnect structure

As can be seen in Figure 1, the CLBs form the central logic structure with easy access to all support and routing structures. The IOBs are located around all the logic and memory elements for easy and quick routing of signals on and off the chip.

Values stored in static memory cells control all the configurable logic elements and interconnect resources. These values load into the memory cells on power-up, and can reload if necessary to change the function of the device.

Each of these elements will be discussed in detail in the following sections.

Input/Output Block

The Spartan-II IOB, as seen in Figure 1, features inputs and outputs that support a wide variety of I/O signaling standards. These high-speed inputs and outputs are capable of supporting various state of the art memory and bus interfaces. Table 1 lists several of the standards which are supported along with the required reference, output and termination voltages needed to meet the standard.

The three IOB registers function either as edge-triggered D-type flip-flops or as level- sensitive latches. Each IOB has a clock signal (CLK) shared by the three registers and independent Clock Enable (CE) signals for each register.

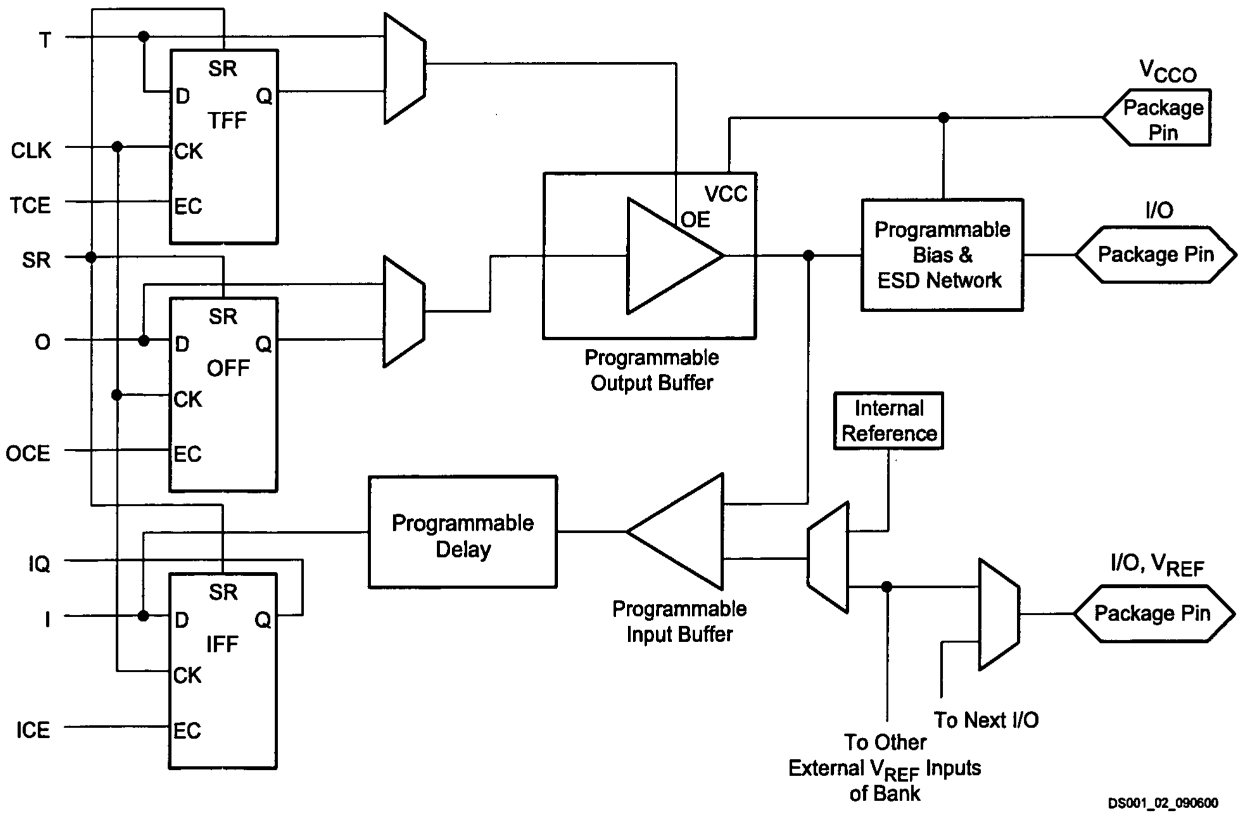


Figure 1: **Spartan-II Input/Output Block (IOB)**

In addition to the CLK and CE control signals, the three registers share a Set/Reset (SR). For each register, this signal can be independently configured as a synchronous Set, a synchronous Reset, an asynchronous Preset, or an asynchronous Clear.

A feature not shown in the block diagram, but controlled by the software, is polarity control. The input and output buffers and all of the IOB control signals have independent polarity controls.

Optional pull-up and pull-down resistors and an optional weak-keeper circuit are attached to each pad. Prior to configuration all outputs not involved in configuration are forced into their high-impedance state. The pull-down resistors and the weak-keeper circuits are inactive, but inputs may optionally be pulled up.

Table 1: **Standards Supported by I/O (Typical Values)**

I/O Standard	Input Reference Voltage (V_{REF})	Output Source Voltage (V_{CCO})	Board Termination Voltage (V_{TT})
LVTTL (2-24 mA)	N/A	3.3	N/A
LVCMOS2	N/A	2.5	N/A
PCI (3V/5V, 33 MHz/66 MHz)	N/A	3.3	N/A
GTL	0.8	N/A	1.2
GTL+	1.0	N/A	1.5
HSTL Class I	0.75	1.5	0.75
HSTL Class III	0.9	1.5	1.5
HSTL Class IV	0.9	1.5	1.5
SSTL3 Class I and II	1.5	3.3	1.5
SSTL2 Class I and II	1.25	2.5	1.25
CTT	1.5	3.3	1.5
AGP-2X	1.32	3.3	N/A

The activation of pull-up resistors prior to configuration is controlled on a global basis by the configuration mode pins. If the pull-up resistors are not activated, all the pins will float. Consequently, external pull-up or pull-down resistors must be provided on pins required to be at a well-defined logic level prior to configuration.

All pads are protected against damage from electrostatic discharge (ESD) and from over-voltage transients. Two forms of over-voltage protection are provided, one that permits 5V compliance, and one that does not. For 5V compliance, a zener-like structure connected to ground turns on when the output rises to approximately 6.5V. When 5V compliance is not required, a conventional clamp diode may be connected to the output supply voltage, V_{CCO}. The type of over-voltage protection can be selected independently for each pad.

All Spartan-II IOBs support IEEE 1149.1-compatible boundary scan testing.

Input Path

A buffer In the Spartan-II IOB input path routes the input signal either directly to internal logic or through an optional input flip-flop.

An optional delay element at the D-input of this flip-flop eliminates pad-to-pad hold time. The delay is matched to the internal clock-distribution delay of the FPGA, and when used, assures that the pad-to-pad hold time is zero.

Each input buffer can be configured to conform to any of the low-voltage signaling standards supported. In some of these standards the input buffer utilizes a user-supplied threshold voltage, V_{REF} The need to supply V_{REF} imposes constraints on which standards can used in close proximity to each other. See **I/O Banking**, page 3.

There are optional pull-up and pull-down resistors at each input for use after configuration.

Output Path

The output path includes a 3-state output buffer that drives the output signal onto the pad. The output signal can be routed to the buffer directly from the internal logic or through an optional IOB output flip-flop.

The 3-state control of the output can also be routed directly from the internal logic or through a flip-flip that provides synchronous enable and disable.

Each output driver can be individually programmed for a wide range of low-voltage signaling standards. Each output buffer can source up to 24 mA and sink up to 48 mA. Drive strength and slew rate controls minimize bus transients.

In most signaling standards, the output high voltage depends on an externally supplied V_{CCO} voltage. The need to supply V_{CCO} imposes constraints on which standards can be used in close proximity to each other. See **I/O Banking**.

An optional weak-keeper circuit is connected to each output. When selected, the circuit monitors the voltage on the pad and weakly drives the pin High or Low to match the input signal. If the pin is connected to a multiple-source signal, the weak keeper holds the signal in its last state if all drivers are disabled. Maintaining a valid logic level in this way helps eliminate bus chatter.

Because the weak-keeper circuit uses the IOB input buffer to monitor the input level, an appropriate V_{REF} voltage must be provided if the signaling standard requires one. The provision of this voltage must comply with the I/O banking rules.

I/O Banking

Some of the I/O standards described above require V_{CCO} and/or V_{REF} voltages. These voltages are externally connected to device pins that serve groups of IOBs, called banks. Consequently, restrictions exist about which I/O standards can be combined within a given bank.

Eight I/O banks result from separating each edge of the FPGA into two banks (see Figure 2). Each bank has multiple V_{CCO} pins which must be connected to the same voltage. Voltage is determined by the output standards in use.

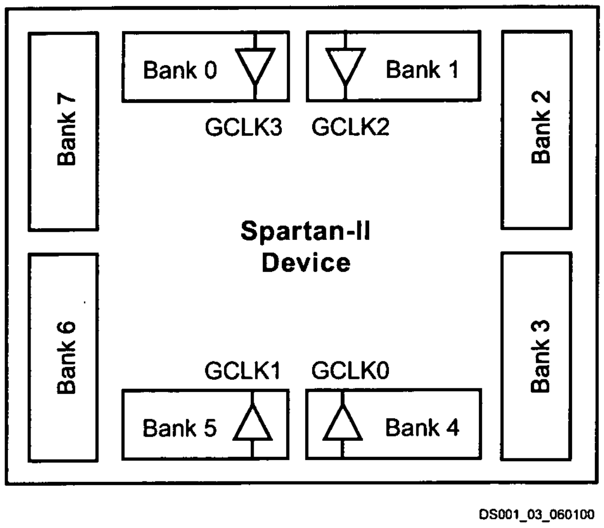


Figure 2: **Spartan-II I/O Banks**

Within a bank, output standards may be mixed only if they use the same V_{CCO}. Compatible standards are shown in Table 2. GTL and GTL+ appear under all voltages because their open-drain outputs do not depend on V_{CCO}.

Table 2: **Compatible Output Standards**

V_{CCO}	Compatible Standards
3.3V	PCI, LVTTL, SSTL3 I, SSTL3 II, CTT, AGP, GTL, GTL+
2.5V	SSTL2 I, SSTL2 II, LVCMOS2, GTL, GTL+
1.5V	HSTL I, HSTL III, HSTL IV, GTL, GTL+

Some input standards require a user-supplied threshold voltage, V_{REF} In this case, certain user-I/O pins are automatically configured as inputs for the V_{REF} voltage. About one in six of the I/O pins in the bank assume this role.

V_{REF} pins within a bank are interconnected internally and consequently only one V_{REF} voltage can be used within each bank. All V_{REF} pins in the bank, however, must be connected to the external voltage source for correct operation.

In a bank, inputs requiring V_{REF} can be mixed with those that do not but only one V_{REF} voltage may be used within a bank. Input buffers that use V_{REF} are not 5V tolerant. LVTTL, LVCMOS2, and PCI are 5V tolerant. The V_{CCO} and V_{REF} pins for each bank appear in the device pinout tables.

Within a given package, the number of V_{REF} and V_{CCO} pins can vary depending on the size of device. In larger devices, more I/O pins convert to V_{REF} pins. Since these are always a superset of the V_{REF} pins used for smaller devices, it is possible to design a PCB that permits migration to a larger device. All V_{REF} pins for the largest device anticipated must be connected to the V_{REF} voltage, and not used for I/O.

Table 3: **Independent Banks Available**

Package	VQ100 PQ208	CS144 TQ144	FG256 FG456
Independent Banks	1	4	8

Configurable Logic Block

The basic building block of the Spartan-II CLB is the logic cell (LC). An LC includes a 4-input function generator, carry logic, and storage element. Output from the function generator in each LC drives the CLB output and the D input of the flip-flop. Each Spartan-II CLB contains four LCs, organized in two similar slices; a single slice is shown in Figure 3.

In addition to the four basic LCs, the Spartan-II CLB contains logic that combines function generators to provide functions of five or six inputs.

Look-Up Tables

Spartan-II function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16 x 1-bit synchronous RAM. Furthermore, the two LUTs within a slice can be combined to create a 16 x 2-bit or 32 x 1-bit synchronous RAM, or a 16 x 1-bit dual-port synchronous RAM.

The Spartan-II LUT can also provide a 16-bit shift register that is ideal for capturing high-speed or burst-mode data. This mode can also be used to store data in applications such as Digital Signal Processing.

Storage Elements

Storage elements in the Spartan-II slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches. The D inputs can be driven either by function generators within the slice or directly from slice inputs, bypassing the function generators.

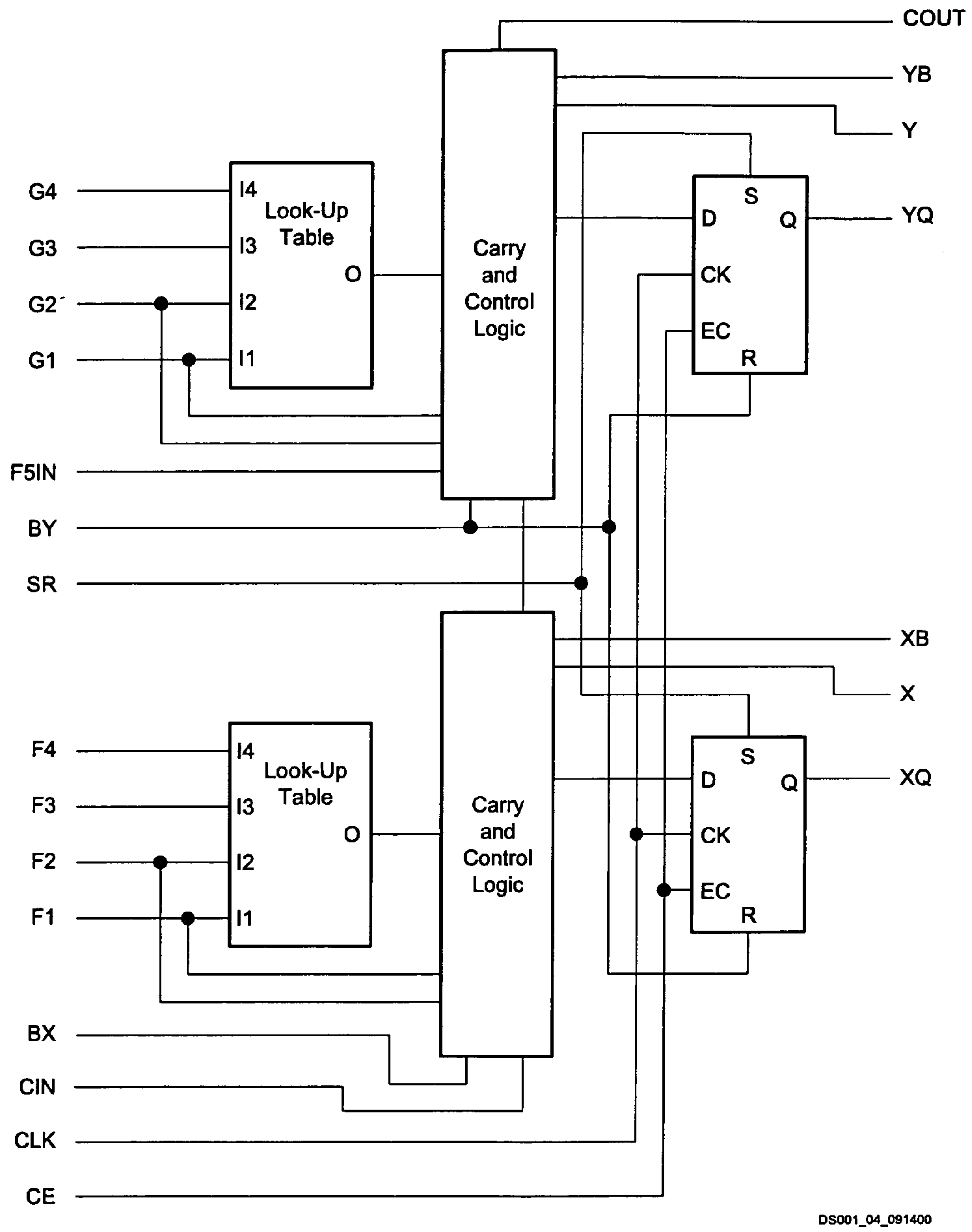


Figure 3: **Spartan-II CLB Slice** (two identical slices in each CLB)

In addition to Clock and Clock Enable signals, each slice has synchronous set and reset signals (SR and BY). SR forces a storage element into the initialization state specified for it in the configuration. BY forces it into the opposite state. Alternatively, these signals may be configured to operate asynchronously.

All control signals are independently invertible, and are shared by the two flip-flops within the slice.

Additional Logic

The F5 multiplexer in each slice combines the function generator outputs. This combination provides either a function generator that can implement any 5-input function, a 4:1 multiplexer, or selected functions of up to nine inputs.

Similarly, the F6 multiplexer combines the outputs of all four function generators in the CLB by selecting one of the F5-multiplexer outputs. This permits the implementation of any 6-input function, an 8:1 multiplexer, or selected functions of up to 19 inputs.

Each CLB has four direct feedthrough paths, one per LC. These paths provide extra data input lines or additional local routing that does not consume logic resources.

Arithmetic Logic

Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. The Spartan-II CLB supports two separate carry chains, one per slice. The height of the carry chains is two bits per CLB.

The arithmetic logic includes an XOR gate that allows a 1-bit full adder to be implemented within an LC. In addition, a dedicated AND gate improves the efficiency of multiplier implementation.

The dedicated carry path can also be used to cascade function generators for implementing wide logic functions.

BUFTs

Each Spartan-II CLB contains two 3-state drivers (BUFTs) that can drive on-chip busses. See **Dedicated Routing**, page 6. Each Spartan-II BUFT has an independent 3-state control pin and an independent input pin.

Block RAM

Spartan-II FPGAs incorporate several large block RAM memories. These complement the distributed RAM Look-Up Tables (LUTs) that provide shallow memory structures implemented in CLBs.

Block RAM memory blocks are organized in columns. All Spartan-II devices contain two such columns, one along each vertical edge. These columns extend the full height of the chip. Each memory block is four CLBs high, and consequently, a Spartan-II device eight CLBs high will contain two memory blocks per column, and a total of four blocks.

Table 4: **Spartan-II Block RAM Amounts**

Spartan-II Device	# of Blocks	Total Block RAM Bits
XC2S15	4	16K
XC2S30	6	24K
XC2S50	8	32K
XC2S100	10	40K
XC2S150	12	48K
XC2S200	14	56K

Each block RAM cell, as illustrated in Figure 4, is a fully synchronous dual-ported 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured independently, providing built-in bus-width conversion.

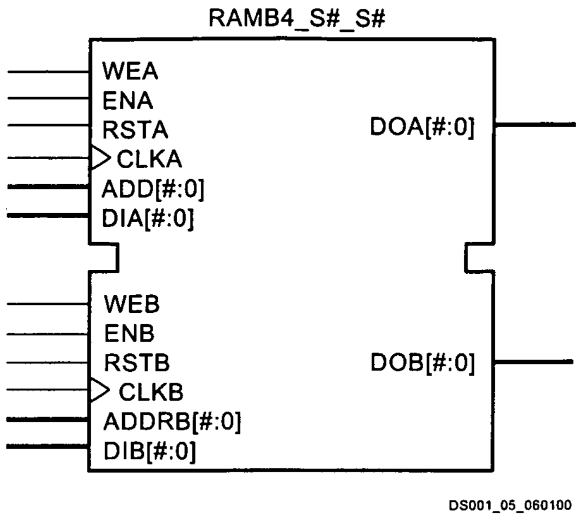


Figure 4: **Dual-Port Block RAM**

Table 5 shows the depth and width aspect ratios for the block RAM.

Table 5: **Block RAM Port Aspect Ratios**

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

The Spartan-II block RAM also includes dedicated routing to provide an efficient interface with both CLBs and other block RAMs.

Programmable Routing Matrix

It is the longest delay path that limits the speed of any worst-case design. Consequently, the Spartan-II routing architecture and its place-and-route software were defined in a single optimization process. This joint optimization minimizes long-path delays, and consequently, yields the best system performance.

The joint optimization also reduces design compilation times because the architecture is software-friendly. Design cycles are correspondingly reduced due to shorter design iteration times.

Local Routing

The local routing resources, as shown in Figure 5, provide the following three types of connections:

- Interconnections among the LUTs, flip-flops, and General Routing Matrix (GRM)
- Internal CLB feedback paths that provide high-speed connections to LUTs within the same CLB, chaining

them together with minimal routing delay

- Direct paths that provide high-speed connections between horizontally adjacent CLBs, eliminating the delay of the GRM

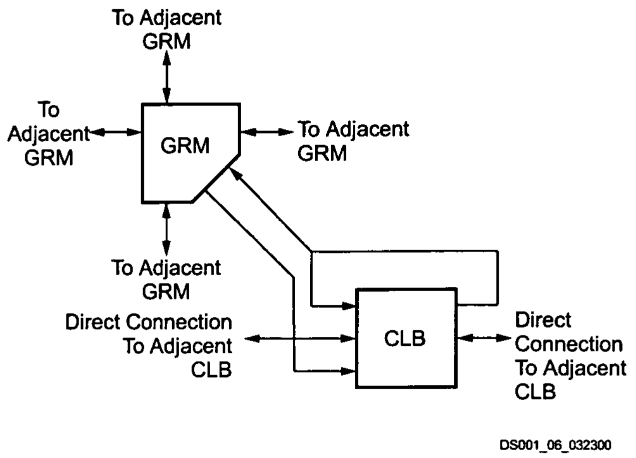


Figure 5: **Spartan-II Local Routing**

General Purpose Routing

Most Spartan-II signals are routed on the general purpose routing, and consequently, the majority of interconnect resources are associated with this level of the routing hierarchy. The general routing resources are located in horizontal and vertical routing channels associated with the rows and columns CLBs. The general-purpose routing resources are listed below.

- Adjacent to each CLB is a General Routing Matrix (GRM). The GRM is the switch matrix through which horizontal and vertical routing resources connect, and is also the means by which the CLB gains access to the general purpose routing.
- 24 single-length lines route GRM signals to adjacent GRMs in each of the four directions.
- 96 buffered Hex lines route GRM signals to other GRMs six blocks away in each one of the four directions. Organized in a staggered pattern, Hex lines may be driven only at their endpoints. Hex-line signals can be accessed either at the endpoints or at the midpoint (three blocks from the source). One third of the Hex lines are bidirectional, while the remaining ones are unidirectional.
- 12 Longlines are buffered, bidirectional wires that distribute signals across the device quickly and efficiently. Vertical Longlines span the full height of the device, and horizontal ones span the full width of the device.

I/O Routing

Spartan-II devices have additional routing resources around their periphery that form an interface between the CLB array and the IOBs. This additional routing, called the VersaRing, facilitates pin-swapping and pin-locking, such that logic redesigns can adapt to existing PCB layouts. Time-to-market is reduced, since PCBs and other system components can be manufactured while the logic design is still in progress.

Dedicated Routing

Some classes of signal require dedicated routing resources to maximize performance. In the Spartan-II architecture, dedicated routing resources are provided for two classes of signal.

- Horizontal routing resources are provided for on-chip 3-state busses. Four partitionable bus lines are provided per CLB row, permitting multiple busses within a row, as shown in Figure 6.
- Two dedicated nets per CLB propagate carry signals vertically to the adjacent CLB.

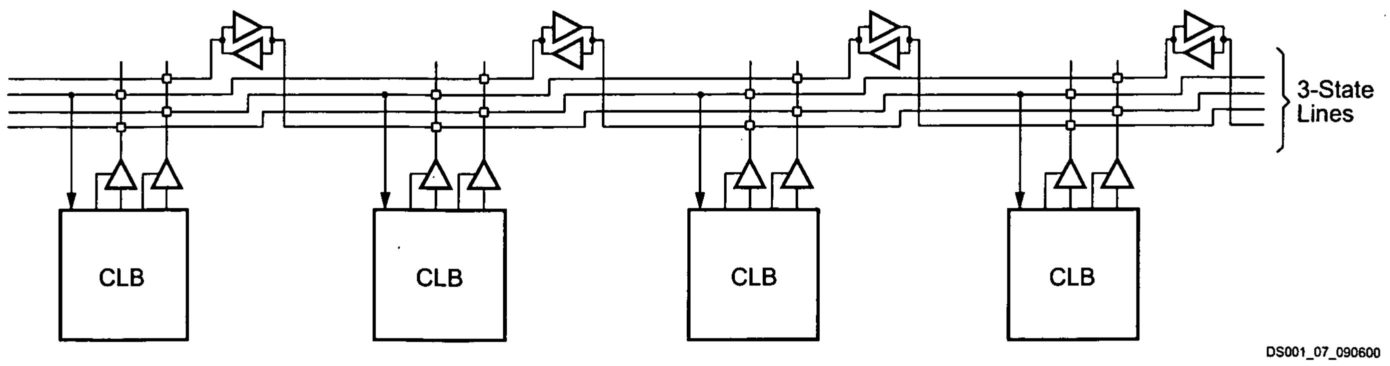


Figure 6: **BUFT Connections to Dedicated Horizontal Bus Lines**

Global Routing

Global Routing resources distribute clocks and other signals with very high fanout throughout the device. Spartan-II devices include two tiers of global routing resources referred to as primary and secondary global routing resources.

- The primary global routing resources are four dedicated global nets with dedicated input pins that are designed to distribute high-fanout clock signals with minimal skew. Each global clock net can drive all CLB, IOB, and block RAM clock pins. The primary global nets may only be driven by global buffers. There are four global buffers, one for each global net.
- The secondary global routing resources consist of 24 backbone lines, 12 across the top of the chip and 12 across bottom. From these lines, up to 12 unique signals per column can be distributed via the 12 longlines in the column. These secondary resources are more flexible than the primary resources since they are not restricted to routing only to clock pins.

Clock Distribution

The Spartan-II family provides high-speed, low-skew clock distribution through the primary global routing resources described above. A typical clock distribution net is shown in Figure 7.

Four global buffers are provided, two at the top center of the device and two at the bottom center. These drive the four primary global nets that in turn drive any clock pin.

Four dedicated clock pads are provided, one adjacent to each of the global buffers. The input to the global buffer is selected either from these pads or from signals in the general purpose routing.

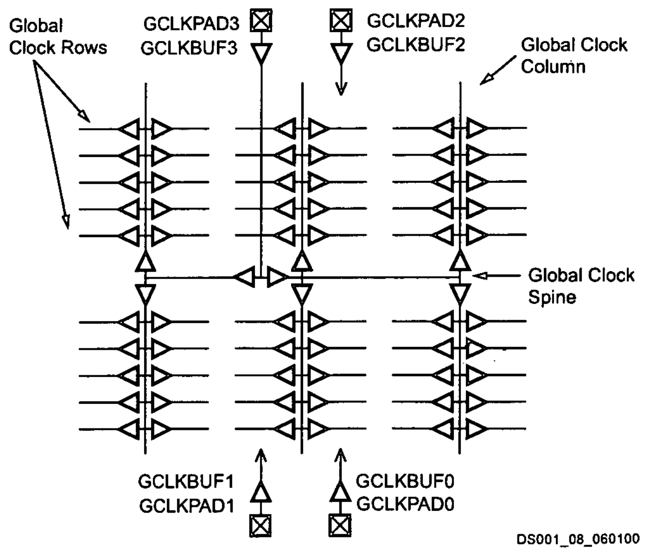


Figure 7: **Global Clock Distribution Network**

Delay-Locked Loop (DLL)

Associated with each global clock input buffer is a fully digital Delay-Locked Loop (DLL) that can eliminate skew between the clock input pad and internal clock-input pins throughout the device. Each DLL can drive two global clock networks. The DLL monitors the input clock and the distributed clock, and automatically adjusts a clock delay element. Additional delay is introduced such that clock edges reach internal flip-flops exactly one clock period after they arrive at the input. This closed-loop system effectively eliminates clock-distribution delay by ensuring that clock edges arrive at internal flip-flops in synchronism with clock edges arriving at the input.

In addition to eliminating clock-distribution delay, the DLL provides advanced control of multiple clock domains. The DLL provides four quadrature phases of the source clock, can double the clock, or divide the clock by 1.5, 2, 2.5, 3, 4, 5, 8, or 16. It has six outputs.

The DLL also operates as a clock mirror. By driving the output from a DLL off-chip and then back on again, the DLL can be used to deskew a board level clock among multiple Spartan-II devices.

In order to guarantee that the system clock is operating correctly prior to the FPGA starting up after configuration, the DLL can delay the completion of the configuration process until after it has achieved lock.

Boundary Scan

Spartan-II devices support all the mandatory boundary-scan instructions specified in the IEEE standard 1149.1. A Test Access Port (TAP) and registers are provided that implement the EXTEST, SAMPLE/PRELOAD, and BYPASS instructions. The TAP also supports two USERCODE instructions and internal scan chains.

The TAP uses dedicated package pins that always operate using LVTTL. For TDO to operate using LVTTL, the V_{CCO} for Bank 2 must be 3.3V. Otherwise, TDO switches rail-to-rail between ground and V_{CCO}.

Boundary-scan operation is independent of individual IOB configurations, and unaffected by package type. All IOBs, including unbonded ones, are treated as independent 3-state bidirectional pins in a single scan chain. Retention of the bidirectional test capability after configuration facilitates the testing of external interconnections.

Table 6 lists the boundary-scan instructions supported in Spartan-II FPGAs. Internal signals can be captured during EXTEST by connecting them to unbonded or unused IOBs. They may also be connected to the unused outputs of IOBs defined as unidirectional input pins. This technique partially compensates for the absence of INTEST support.

Table 6: **Boundary-Scan Instructions**

Boundary-Scan Command	Binary Code[4:0]	Description
EXTEST	00000	Enables boundary-scan EXTEST operation
SAMPLE	00001	Enables boundary-scan SAMPLE operation
USR1	00010	Access user-defined register 1
USR2	00011	Access user-defined register 2
CFG_OUT	00100	Access the configuration bus for Readback
CFG_IN	00101	Access the configuration bus for Configuration
INTEST	00111	Enables boundary-scan INTEST operation
USRCODE	01000	Enables shifting out USER code
IDCODE	01001	Enables shifting out of ID Code
HIZ	01010	Disables output pins while enabling the Bypass Register
JSTART	01100	Clock the start-up sequence when StartupClk is TCK
BYPASS	11111	Enables BYPASS
RESERVED	All other codes	Xilinx reserved instructions

The public boundary-scan instructions are available prior to configuration. After configuration, the public instructions remain available together with any USERCODE instructions installed during the configuration. While the SAMPLE and BYPASS instructions are available during configuration, it is recommended that boundary-scan operations not be performed during this transitional period.

In addition to the test instructions outlined above, the boundary-scan circuitry can be used to configure the FPGA, and also to read back the configuration data.

To facilitate internal scan chains, the User Register provides three outputs (Reset, Update, and Shift) that represent the corresponding states in the boundary-scan internal state machine.

Figure 8 is a diagram of the Spartan-II family boundary scan logic. It includes three bits of Data Register per IOB, the IEEE 1149.1 Test Access Port controller, and the Instruction Register with decodes.

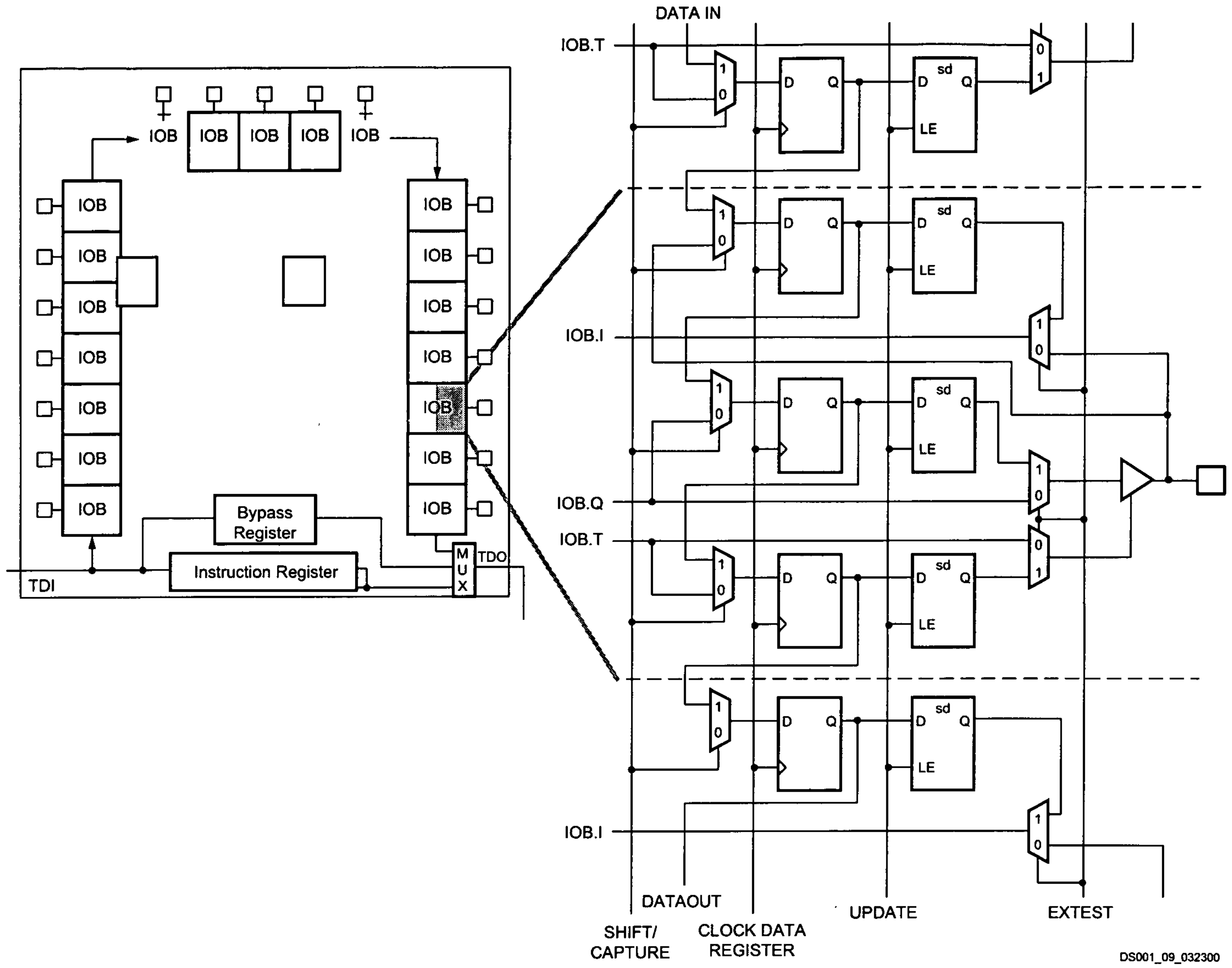


Figure 8: **Spartan-II Family Boundary Scan Logic**

Bit Sequence

The bit sequence within each IOB is: In, Out, 3-State. The input-only pins contribute only the In bit to the boundary scan I/O data register, while the output-only pins contributes all three bits.

From a cavity-up view of the chip (as shown in the FPGA Editor), starting in the upper right chip corner, the boundary scan data-register bits are ordered as shown in Figure 9.

BSDL (Boundary Scan Description Language) files for Spartan-II family devices are available on the Xilinx website, in the File Download area.

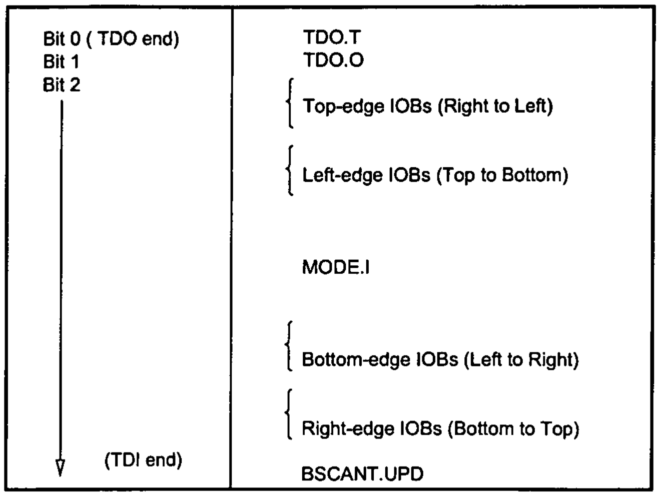

Bit 0 (TDO end)
Bit 1
Bit 2
(TDI end)
TDO.T
TDO.O
Top-edge IOBs (Right to Left)
Left-edge IOBs (Top to Bottom)
MODE.I
Bottom-edge IOBs (Left to Right)
Right-edge IOBs (Bottom to Top)
BSCANT.UPD
DS001_10_032300

Figure 9: **Boundary Scan Bit Sequence**

Development System

Spartan-II FPGAs are supported by the Xilinx Foundation and Alliance CAE tools. The basic methodology for Spartan-II design consists of three interrelated steps: design entry, implementation, and verification. Industry-standard tools are used for design entry and simulation (for example, Synopsys FPGA Express), while Xilinx provides proprietary architecture-specific tools for implementation.

The Xilinx development system is integrated under the Xilinx Design Manager software, providing designers with a common user interface regardless of their choice of entry and verification tools. The software simplifies the selection of implementation options with pull-down menus and on-line help.

Application programs ranging from schematic capture to Placement and Routing (PAR) can be accessed through the software. The program command sequence is generated prior to execution, and stored for documentation.

Several advanced software features facilitate Spartan-II design. Relationally-Placed Macros (RPMs), for example, are schematic-based macros with relative location constraints to guide their placement. They help ensure optimal implementation of common functions.

For HDL design entry, the Xilinx FPGA development system provides interfaces to several synthesis design environments.

A standard interface-file specification, Electronic Design Interchange Format (EDIF), simplifies file transfers into and out of the development system.

Spartan-II FPGAs supported by a unified library of standard functions. This library contains over 400 primitives and macros, ranging from 2-input AND gates to 16-bit accumulators, and includes arithmetic functions, comparators, counters, data registers, decoders, encoders, I/O functions, latches, Boolean functions, multiplexers, shift registers, and barrel shifters.

The "soft macro" portion of the library contains detailed descriptions of common logic functions, but does not contain any partitioning or placement information. The performance of these macros depends, therefore, on the partitioning and placement obtained during implementation.

RPMs, on the other hand, do contain predetermined partitioning and placement information that permits optimal implementation of these functions. Users can create their own library of soft macros or RPMs based on the macros and primitives in the standard library.

The design environment supports hierarchical design entry, with high-level schematics that comprise major functional blocks, while lower-level schematics define the logic in these blocks. These hierarchical design elements are automatically combined by the implementation tools. Different design entry tools can be combined within a hierarchical design, thus allowing the most convenient entry method to be used for each portion of the design.

Design Implementation

The place-and-route tools (PAR) automatically provide the implementation flow described in this section. The partitioner takes the EDIF netlist for the design and maps the logic into the architectural resources of the FPGA (CLBs and IOBs, for example). The placer then determines the best locations for these blocks based on their interconnections and the desired performance. Finally, the router interconnects the blocks.

The PAR algorithms support fully automatic implementation of most designs. For demanding applications, however, the user can exercise various degrees of control over the process. User partitioning, placement, and routing information is optionally specified during the design-entry process. The implementation of highly structured designs can benefit greatly from basic floorplanning.

The implementation software incorporates Timing Wizard® timing-driven placement and routing. Designers specify timing requirements along entire paths during design entry. The timing path analysis routines in PAR then recognize these user-specified requirements and accommodate them.

Timing requirements are entered on a schematic in a form directly relating to the system requirements, such as the targeted clock frequency, or the maximum allowable delay between two registers. In this way, the overall performance of the system along entire signal paths is automatically tailored to user-generated specifications. Specific timing information for individual nets is unnecessary.

Design Verification

In addition to conventional software simulation, FPGA users can use in-circuit debugging techniques. Because Xilinx devices are infinitely reprogrammable, designs can be verified in real time without the need for extensive sets of software simulation vectors.

The development system supports both software simulation and in-circuit debugging techniques. For simulation, the system extracts the post-layout timing information from the design database, and back-annotates this information into the netlist for use by the simulator. Alternatively, the user can verify timing-critical portions of the design using the static timing analyzer.

For in-circuit debugging, the development system includes a download and readback cable, which connects the FPGA in the target system to a PC or workstation. After downloading the design into the FPGA, the designer can single-step the logic, readback the contents of the flip-flops, and so observe the internal logic state. Simple modifications can be downloaded into the system in a matter of minutes.

Configuration

Configuration is the process by which the bitstream of a design, as generated by the Xilinx development software, is loaded into the internal configuration memory of the FPGA. Spartan-II devices support both serial configuration, using the master/slave serial and JTAG modes, as well as byte-wide configuration employing the Slave Parallel mode.

Configuration File

Spartan-II devices are configured by sequentially loading frames of data that have been concatenated into a configuration file. Table 7 shows how much nonvolatile storage space is needed for Spartan-II devices.

It is important to note that, while a PROM is commonly used to store configuration data before loading them into the FPGA, it is by no means required. Any of a number of different kinds of under populated nonvolatile storage already available either on or off the board (i.e., hard drives, FLASH cards, etc.) can be used. For more information on configuration without a PROM, refer to **XAPP098**, ***The Low-Cost, Efficient Serial Configuration of Spartan FPGAs***.

Table 7: **Spartan-II Configuration File Size**

Device	Configuration File Size (Bits)
XC2S15	197,696
XC2S30	336,768
XC2S50	559,200
XC2S100	781,216
XC2S150	1,040,096
XC2S200	1,335,840

Modes

Spartan-II devices support the following four configuration modes:

- Slave Serial mode
- Master Serial mode
- Slave Parallel mode
- Boundary-scan mode

The Configuration mode pins (M2, M1, M0) select among these configuration modes with the option in each case of having the IOB pins either pulled up or left floating prior to configuration. The selection codes are listed in Table 8.

Configuration through the boundary-scan port is always available, independent of the mode selection. Selecting the boundary-scan mode simply turns off the other modes. The three mode pins have internal pull-up resistors, and default to a logic High if left unconnected.

Table 8: **Configuration Modes**

Configuration Mode	Preconfiguration Pull-ups	M0	M1	M2	CCLK Direction	Data Width	Serial D_{OUT}
Master Serial mode	No	0	0	0	Out	1	Yes
	Yes	0	0	1			
Slave Parallel mode	Yes	0	1	0	In	8	No
	No	0	1	1			
Boundary-Scan mode	Yes	1	0	0	N/A	1	No
	No	1	0	1			
Slave Serial mode	Yes	1	1	0	In	1	Yes
	No	1	1	1			

Signals

There are two kinds of pins that are used to configure Spartan-II devices: Dedicated pins perform only specific configuration-related functions; the other pins can serve as general purpose I/Os once user operation has begun.

The dedicated pins comprise the mode pins (M2, M1, M0), the configuration clock pin (CCLK), the $\overline{\text{PROGRAM}}$ pin, the DONE pin and the boundary-scan pins (TDI, TDO, TMS, TCK). Depending on the selected configuration mode, CCLK may be an output generated by the FPGA, or may be generated externally, and provided to the FPGA as an input.

Note that some configuration pins can act as outputs. For correct operation, these pins require a V_{CCO} of 3.3V to drive an LVTTL signal. All the relevant pins fall in banks 2 or 3.

For a more detailed description than that given below, see **DS001-4, *Spartan-II 2.5V FPGA Family: Pinout Tables*** and **XAPP176, *Spartan-II FPGA Series Configuration and Readback.***

The Process

The sequence of steps necessary to configure Spartan-II devices are shown in Figure 10. The overall flow can be divided into three different phases.

- Initiating Configuration
- Configuration memory clear
- Loading data frames
- Start-up

The memory clearing and start-up phases are the same for all configuration modes; however, the steps for the loading of data frames are different. Thus, the details for data frame loading are described separately in the sections devoted to each mode.

Initiating Configuration

There are two different ways to initiate the configuration process: applying power to the device or asserting the $\overline{\text{PROGRAM}}$ input.

Configuration on power-up occurs automatically unless it is delayed by the user, as described in a separate section below. The waveform for configuration on power-up is shown in Figure 11, page 13. Before configuration can begin, V_{CCO} Bank 2 must be greater than 1.0V. Furthermore, all V_{CCINT} power pins must be connected to a 2.5V supply. For more information on delaying configuration, see **Clearing Configuration Memory**, page 13.

Once in user operation, the device can be re-configured simply by pulling the $\overline{\text{PROGRAM}}$ pin Low. The device acknowledges the beginning of the configuration process by driving DONE Low, then enters the memory-clearing phase.

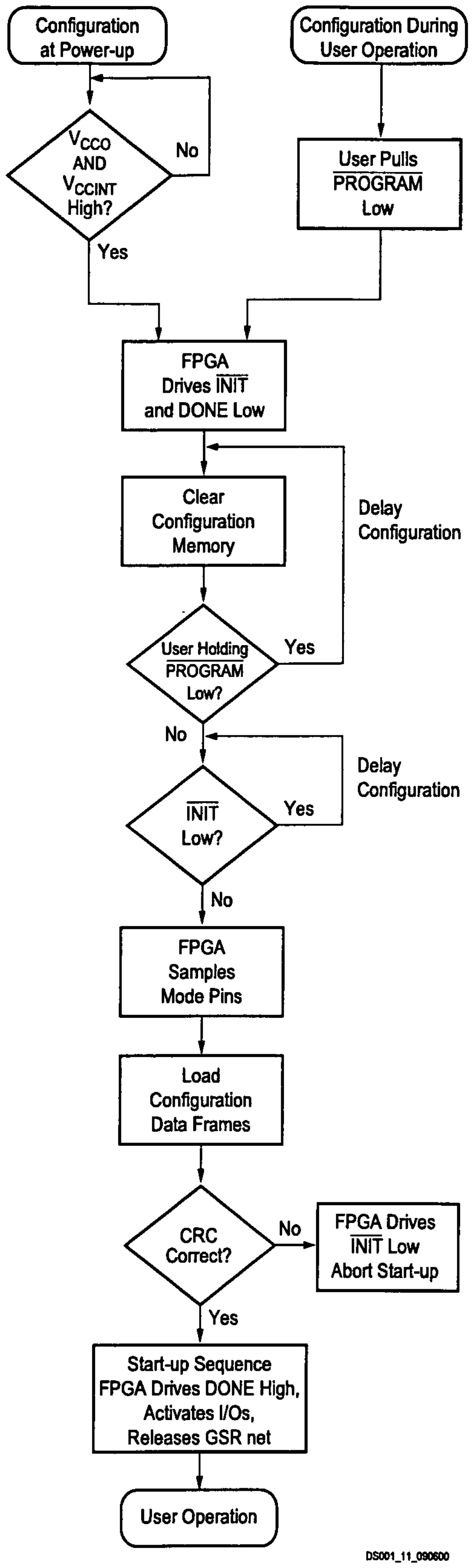
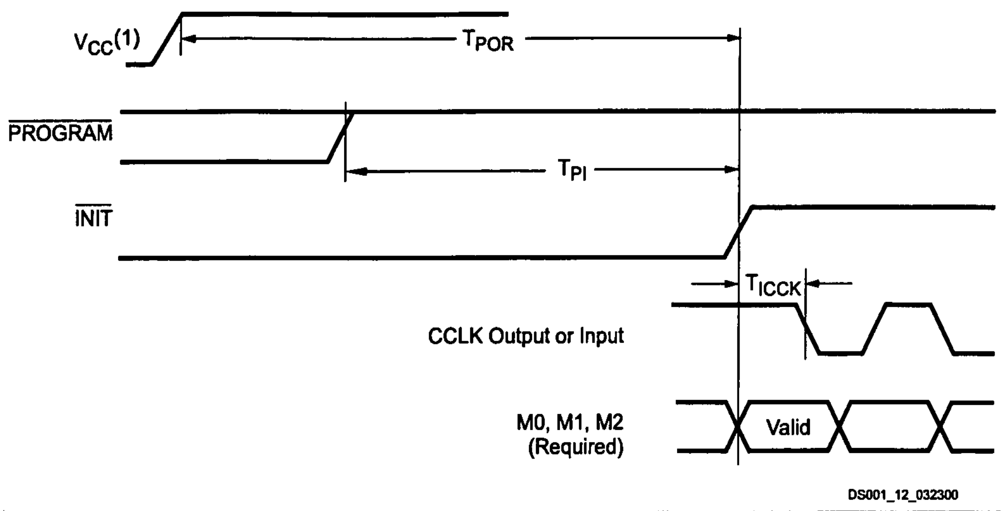

Configuration at Power-up
Configuration During User Operation
VCCO AND VCCINT High?
No
Yes
User Pulls PROGRAM Low
FPGA Drives INIT and DONE Low
Clear Configuration Memory
Delay Configuration
User Holding PROGRAM Low?
Yes
No
INIT Low?
Yes
Delay Configuration
No
FPGA Samples Mode Pins
Load Configuration Data Frames
CRC Correct?
No
FPGA Drives INIT Low Abort Start-up
Yes
Start-up Sequence FPGA Drives DONE High, Activates I/Os, Releases GSR net
User Operation
DS001_11_090600

Figure 10: **Configuration Flow Diagram**



Symbol	Description		Units
T_{POR}	Power-on reset	2	ms, max
T_{PL}	Program latency	100	µs, max
T_{ICCK}	CCLK output delay (Master Serial mode only)	0.5	µs, min
		4	µs, max
$T_{PROGRAM}$	Program pulse width	300	ns, min

Notes: referring to waveform above:

1. Before configuration can begin, V_{CCINT} must be greater than 1.6V and V_{CCO} Bank 2 must be greater than 1.0V.

Figure 11: **Configuration Timing on Power-Up**

Clearing Configuration Memory

The device indicates that clearing the configuration memory is in progress by driving $\overline{\text{INIT}}$ Low. At this time, the user can delay configuration by holding either $\overline{\text{PROGRAM}}$ or $\overline{\text{INIT}}$ Low, which causes the device to remain in the memory clearing phase. Note that the bidirectional $\overline{\text{INIT}}$ line is driving a Low logic level during memory clearing. Thus, to avoid contention, use an open-drain driver to keep $\overline{\text{INIT}}$ Low.

With no delay in force, the device indicates that the memory is completely clear by driving $\overline{\text{INIT}}$ High. The FPGA samples its mode pins on this Low-to-High transition.

Loading Configuration Data

Once $\overline{\text{INIT}}$ is High, the user can begin loading configuration data frames into the device. The details of loading the configuration data are discussed in the sections treating the configuration modes individually. The sequence of operations necessary to load configuration data using the serial modes is shown in Figure 13. Loading data using the Slave Parallel mode is shown in Figure 18, page 18.

CRC Error Checking

During the loading of configuration data, a CRC value embedded in the configuration file is checked against a CRC value calculated within the FPGA. If the CRC values do not match, the FPGA drives $\overline{\text{INIT}}$ Low to indicate that a frame error has occurred and configuration is aborted.

To reconfigure the device, the $\overline{\text{PROGRAM}}$ pin should be asserted to reset the configuration logic. Recycling power also resets the FPGA for configuration. See **Clearing Configuration Memory**.

Start-up

The start-up sequence oversees the transition of the FPGA from the configuration state to full user operation. A match of CRC values, indicating a successful loading of the configuration data, initiates the sequence.

During start-up, the device performs four operations:

1. The assertion of DONE. The failure of DONE to go High may indicate the unsuccessful loading of configuration data.
2. The release of the Global Three State. This activates all the I/Os.
3. Negates Global Set Reset (GSR). This allows all flip-flops to change state.
4. The assertion of Global Write Enable (GWE). This allows all RAMs and flip-flops to change state.

By default, these operations are synchronized to CCLK. The entire start-up sequence lasts eight cycles, called C0-C7, after which the loaded design is fully functional. The default timing for start-up is shown in the top half of Figure 12. The four operations can be selected to switch on any CCLK cycle C1-C6 through settings in the Xilinx Development Software. Heavy lines show default settings.

The bottom half of Figure 12 shows another commonly used version of the start-up timing known as Sync-to-DONE. This version makes the GTS, GSR, and GWE events conditional upon the DONE pin going High. This timing is important for a daisy chain of multiple FPGAs in serial mode, since it ensures that all FPGAs go through start-up together, after all their DONE pins have gone High.

Sync-to-DONE timing is selected by setting the GTS, GSR, and GWE cycles to a value of DONE in the configuration options. This causes these signals to transition one clock cycle after DONE externally transitions High.

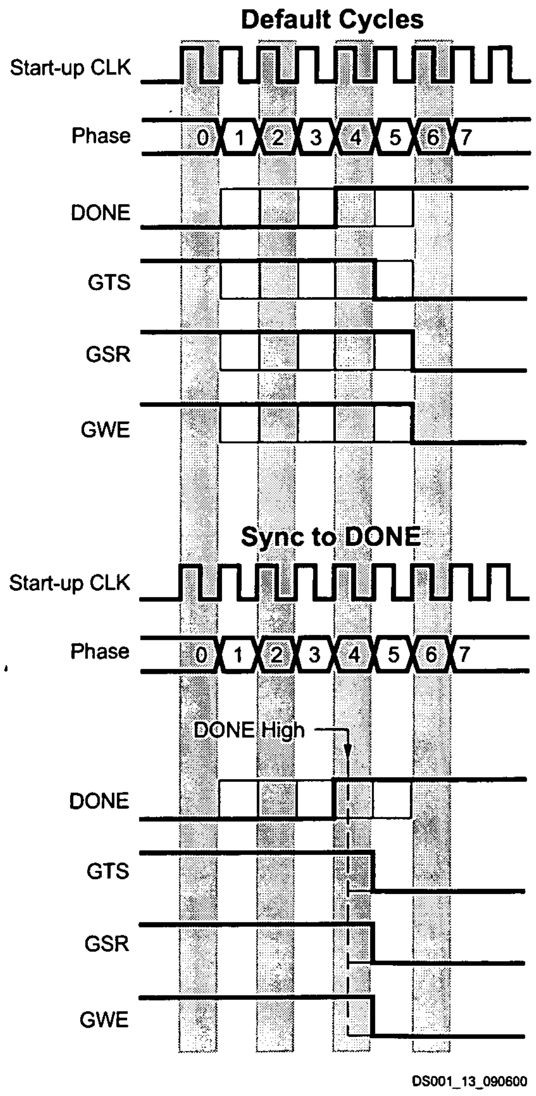


Figure 12: **Start-Up Waveforms**

Serial Modes

There are two serial configuration modes: In Master Serial mode, the FPGA controls the configuration process by driving CCLK as an output. In Slave Serial mode, the FPGA passively receives CCLK as an input from an external agent (e.g., a microprocessor, CPLD, or second FPGA in master mode) that is controlling the configuration process. In both modes, the FPGA is configured by loading one bit per CCLK cycle. The MSB of each configuration data byte is always written to the DIN pin first.

See Figure 13 for the sequence for loading data into the Spartan-II FPGA serially. This is an expansion of the "Load Configuration Data Frames" block in Figure 10, page 12.

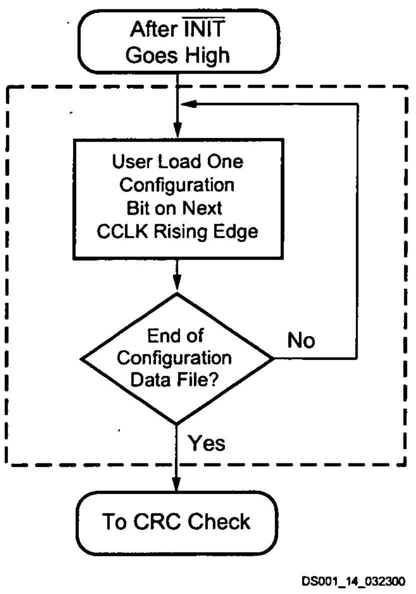
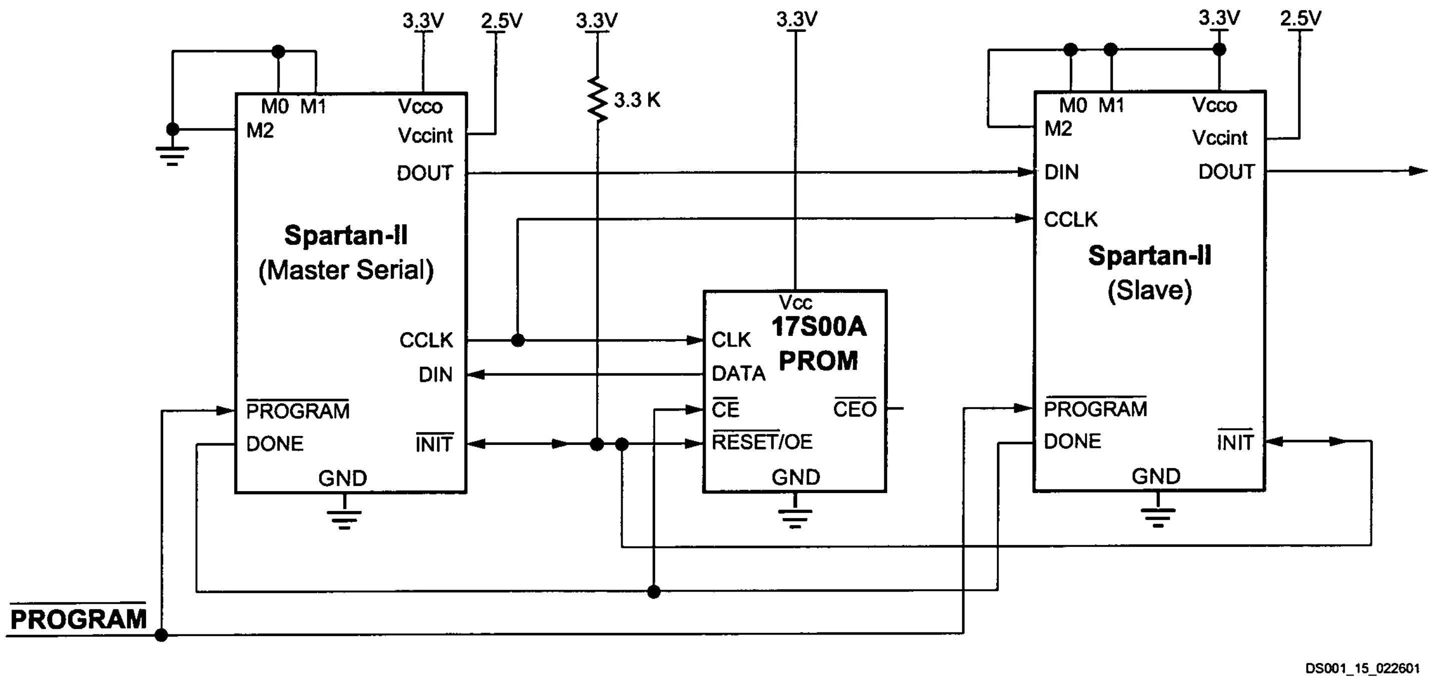

After $\overline{\text{INIT}}$ Goes High
User Load One Configuration Bit on Next CCLK Rising Edge
End of Configuration Data File?
No
Yes
To CRC Check
DS001_14_032300

Figure 13: **Loading Serial Mode Configuration Data**

Slave Serial Mode

In Slave Serial mode, the FPGAs CCLK pin is driven by an external source, allowing FPGAs to be configured from other logic devices such as microprocessors or in a daisy-chain configuration. Figure 14 shows connections for a Master Serial FPGA configuring a Slave Serial FPGA from a PROM. A Spartan-II device in slave serial mode should be connected as shown for the third device from the left. Slave Serial mode is selected by a <11x> on the mode pins (M0, M1, M2).

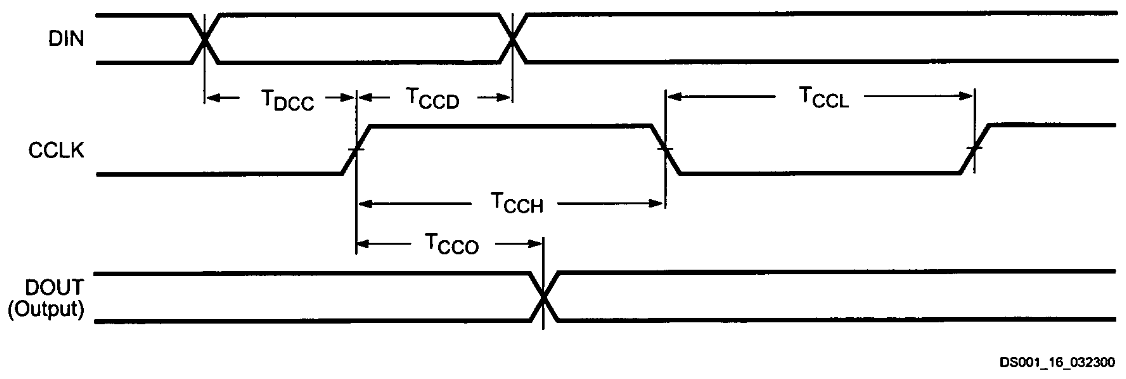
Figure 15 shows the timing for Slave Serial configuration. The serial bitstream must be setup at the DIN input pin a short time before each rising edge of an externally generated CCLK. Multiple FPGAs in Slave Serial mode can be daisy-chained for configuration from a single source. After an FPGA is configured, data for the next device is routed to the DOUT pin. Data on the DOUT pin changes on the rising edge of CCLK. Configuration must be delayed until $\overline{\text{INIT}}$ pins of all daisy-chained FPGAs are High. For more information, see **Start-up**, page 13.



Notes:

1. If the DriveDone configuration option is not active for any of the FPGAs, pull up DONE with a 3.3K Ω resistor.

Figure 14: **Master/Slave Serial Configuration Circuit Diagram**



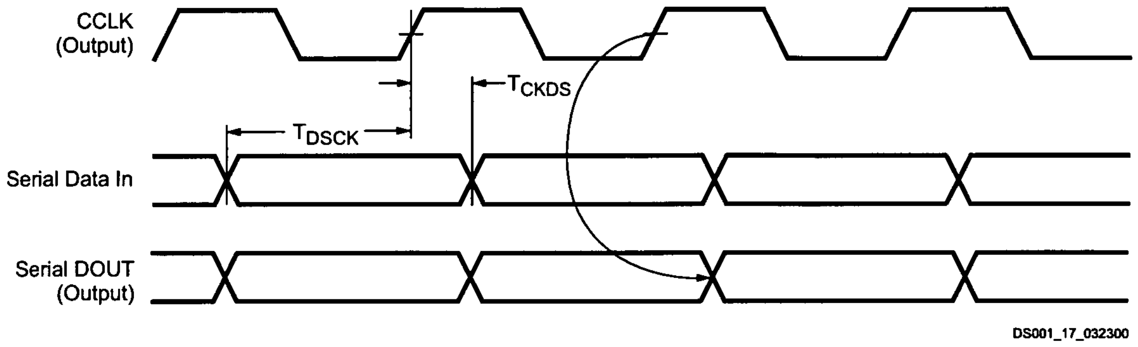
Symbol		Description		Units
T_{DCC}	CCLK	DIN setup	5	ns, min
T_{CCD}		DIN hold	0	ns, min
T_{CCO}		DOUT	12	ns, max
T_{CCH}		High time	5	ns, min
T_{CCL}		Low time	5	ns, min
F_{CC}		Maximum frequency	66	MHz, max

Figure 15: **Slave Serial Mode Timing**

Master Serial Mode

In Master Serial mode, the CCLK output of the FPGA drives a Xilinx PROM which feeds a serial stream of configuration data to the FPGA's DIN input. Figure 14 shows a Master Serial FPGA configuring a Slave Serial FPGA from a PROM. A Spartan-II device in Master Serial mode should be connected as shown for the device on the left side. Master Serial mode is selected by a <00x> on the mode pins (M0, M1, M2). The PROM RESET pin is driven by $\overline{\text{INIT}}$, and CE input is driven by DONE. The interface is identical to the slave serial mode except that an oscillator internal to the FPGA is used to generate the configuration clock (CCLK). Any of a number of different frequencies ranging from 4 to 60 MHz can be set using the ConfigRate option in the Xilinx development software. On power-up, while the first 60 bytes of the configuration data are being loaded, the CCLK frequency is always 2.5 MHz. This frequency is used until the ConfigRate bits, part of the configuration file, have been loaded into the FPGA, at which point, the frequency changes to the selected ConfigRate. Unless a different frequency is specified in the design, the default ConfigRate is 4 MHz. The period of the CCLK signal created by the internal oscillator has a variance of +45%, –30% from the specified value.

Figure 16 shows the timing for Master Serial configuration. The FPGA accepts one bit of configuration data on each rising CCLK edge. After the FPGA has been loaded, the data for the next device in a daisy-chain is presented on the DOUT pin after the rising CCLK edge.



Symbol		Description		Units
T_{DSCK}	CCLK	DIN setup	5.0	ns, min
T_{CKDS}		DIN hold	0.0	ns, min
		Frequency tolerance with respect to nominal	+45%, –30%	-

Figure 16: **Master Serial Mode Timing**

Slave Parallel Mode

The Slave Parallel mode is the fastest configuration option. Byte-wide data is written into the FPGA. A BUSY flag is provided for controlling the flow of data at a clock frequency F_{CCNH} above 50 MHz.

Figure 17, page 17 shows the connections for two Spartan-II devices using the Slave Parallel mode. Slave Parallel mode is selected by a <011> on the mode pins (M0, M1, M2).

The agent controlling configuration is not shown. Typically, a processor, a microcontroller, or CPLD controls the Slave Parallel interface. The controlling agent provides byte-wide configuration data, CCLK, a Chip Select ($\overline{\text{CS}}$) signal and a Write signal ($\overline{\text{WRITE}}$). If BUSY is asserted (High) by the FPGA, the data must be held until BUSY goes Low.

After configuration, the pins of the Slave Parallel port (D0-D7) can be used as additional user I/O. Alternatively, the port may be retained to permit high-speed 8-bit readback. Then data can be read by de-asserting $\overline{\text{WRITE}}$. See **Readback**, page 18.